Algorithms
Computer Science 140 & Mathematics 168
Spring 2012
Homework 1b
Due Tuesday, January 24

- Please use LaTeX to typeset your solutions to this assignment.
- **Please remember to submit each problem on a separate sheet of paper with your name on the top.** (If a problem requires multiple sheets, please staple them together).
- About bonus problems: Please take a look at the bonus problems when they are given. If you have the time and inclination, work on them! Working on these problems will give you a deeper understanding of the material. Moreover, the extra points on these problems are potentially substantial (bonus points have the same value as regular homework points).

1. [**15 points**] Indicate whether each of the following statements is true or false and then carefully **prove** your answer using the formal definition of Big-Oh, Big-Omega, and Big-Theta notation and properties of logarithms from Homework 1a. Remember that to show that one of these statements is false, you must obtain a formal contradiction; it does not suffice to just say "false".

   (a) $2^{n+1} \in O(2^n)$.

   (b) $2^{2n} \in O(2^n)$.

   (c) $3n^2 \log_2 n + 16n \in O(n^3)$.

   (d) $25 \log_2 8n^{10} \in O(\log_{10} n)$.

   (e) $4^{\log_2 n} \in O(n^3)$.

   (f) $5n^2 + 42 \in \Omega(n)$.

   (g) $6n^3 + 26n \in \Theta(n^3)$.

2. [**25 Points**] **Curly, Mo, and Larry's Totally Excellent (?) Sorting Algorithm!** Professors Curly, Mo, and Larry of the Pasadena Institute of Technology have proposed the following sorting algorithm: First sort the first two-thirds of the elements in the array. Next sort the last two thirds of the array. Finally, sort the first two thirds again. (Notice that this algorithm is similar to Mergesort except that it uses three recursive calls rather than two

and there is no merging step! As a consequence, this algorithm is very easy to implement!) The code is given below. Notice that the floor function, $\lfloor x \rfloor$, simply rounds down to the nearest integer. This is just used to compute the appropriate two-thirds and round to an integer so that we don't use non-integer indices into our array!

```
Stooge-sort (A,i,j)
begin
    if A[i] > A[j] then
        swap A[i] and A[j]
    if i + 1 ≥ j then
        return
    k = ⌊(j − i + 1)/3⌋.
    Stooge-sort(A, i, j − k)      Comment: Sort first two-thirds.
    Stooge-sort(A, i + k, j)      Comment: Sort last two-thirds.
    Stooge-sort(A, i, j − k)      Comment: Sort first two-thirds again!
end
```

(a) Give an informal but convincing explanation (not a rigorous proof by induction) of why the approach of sorting the first two-thirds of the array, then sorting the last two-thirds of the array, and then sorting again the first two-thirds of the array yields a sorted array. A few well-chosen sentences should suffice here.

(b) Find a recurrence relation for the worst-case running time of Stooge-sort. To simplify your recurrence relation, you may assume each of the recursive calls is on a portion of the array that is *exactly* two-thirds the length of the original array.

(c) Next, solve the recurrence relation using the work tree method. **Show all of your work.** In your analysis, it will be convenient to choose n to be c^k for some fixed constant c. (For example, we used $c = 2$ when analyzing Mergesort. Here you will want to use a different value of c. The value of c that you choose might not even be an integer! As we've seen in class, this is valid and allows us to significantly simplify the analysis!)

(d) How does the worst-case running time of Stooge-Sort compare with the worst-case running times of the other sorting algorithms that we've seen so far?

3. **[20 Points] Stock Ratings!** A remarkable number of algorithms experts work on Wall Street and in other sectors of the finance industry. Throughout this semester, we'll see a number of applications of algorithms to finance.

Here's the gratuitous backstory for this problem: You've been hired as the Director of Algorithm Design by the brokerage firm of Weil, Proffet, and Howe (WPH). The firm specializes in rating stocks. Their measure of the quality of a stock is the longest consecutive number of days in which the stock did not decrease in value. For example, consider the stock values below:

Day:	1	2	3	4	5	6	7	8
Value:	42	40	45	45	44	43	50	49

In this example, the length of the longest consecutive non-decreasing run is 3. This run goes from day 2 to day 4. (In some cases there could be ties for the longest run, but that's OK! We are only looking for the length of the longest run, not the actual run itself.)

Here are your tasks:

(a) Your predecessor at WPH proposed the following algorithm for stock rating: Using a for loop, iterate through each element in the array. For each such element, use another (nested) for loop to find the longest run from that point forward. Keep track of the longest run that you've found and return that in the end. Show that for some input, the algorithm can use $\Theta(n^2)$ running time.

(b) Next, use the divide-and-conquer paradigm to devise an algorithm whose running time is asymptotically better than $\Theta(n^2)$. Provide pseudo-code (the syntax is up to you, as long as it is clear to any computationally literate reader) or a very clear English description of your algorithm. (*Note: There are algorithms that are fast and do not use divide-and-conquer. Please use divide-and-conquer here though! That is, divide the array into two halves and use recursion on these two halves to help solve the problem. There are several different divide-and-conquer algorithms for this problem. Some are asymptotically faster than n^2 and some are asymptotically MUCH faster than n^2. Any algorithm asymptotically faster than n^2 suffices here, but if you find an algorithm **and** can prove that your algorithm is asymptotically optimal, then you will receive 5 extra bonus points. It's not too hard! Try it!)*

(c) Assuming that n is a power of 2, give a recurrence relation that describes the worst-case running time of your algorithm.

(d) Solve your recurrence relation – using the work tree method that we've used in class – to get a simple closed form for the running time of your

algorithm and then give the asymptotic (Θ notation) running time. Show your analysis in detail. If you want to use a picture to help explain the running time, you may just leave some space in your document and draw it by hand or embed a figure that you've drawn in another tool. Download the source for this document from the course website and read the embedded comments on how to do all this in LaTeX.

(e) Explain briefly why the asymptotic running time still holds even if n is not a power of 2.

(f) Prove the correctness of your algorithm using mathematical induction.

4. **[20 Point Optional Bonus Problem] Chip Testing!** *This is a fun and interesting problem! It is highly recommended!*

Professor Chip Testor has n supposedly identical computer chips that in principle are capable of testing each other. The professor's test jig accommodates two chips at a time. When the jig is loaded, each chip tests the other and reports whether it is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the answer of a bad chip cannot be trusted. Thus, the four possible outcomes of a test are as follows:

Chip A says	Chip B says	Conclusion
B is good	A is good	Both good, or both bad
B is good	A is bad	At least one is bad
B is bad	A is good	At least one is bad
B is bad	A is bad	At least one is bad

(a) Consider the problem of finding a single good chip from among n chips, assuming that more than $\frac{n}{2}$ of the chips are good. Show that $\lfloor \frac{n}{2} \rfloor$ pairwise tests are sufficient to reduce the problem to one of nearly half the size. (You might find it convenient to begin by assuming that n is even. The case that n is odd is just a bit more involved.)

(b) Using the result of part (a) above, show that the good chips can be identified with $\Theta(n)$ pairwise tests, assuming that more than $\frac{n}{2}$ of the chips are good. Explain clearly why your algorithm identifies all of the good chips. Then give and solve the recurrence that describes the number of tests.